

2. cvičení

Datové struktury I, 7. 10. 2024

<https://iuuk.mff.cuni.cz/~chmel/2425/ds1/>

Úloha 1 (Natahujeme pole)

Ukázali jsme si, jak implementovat natahovací pole, které funguje v amortizovaně konstantním čase, s tím, že vždycky, když je pole plné, a potřebujeme přidat další prvek, všechno překopírujeme do pole dvojnásobné velikosti.

Co by se stalo, kdybychom místo zdvojnásobování kapacitu C zvyšovali jinak?

1. $C \rightsquigarrow C + k$, kde $k \geq 1$ je konstanta,
2. $C \rightsquigarrow C^2$,
3. $C \rightsquigarrow k \cdot C$ pro konstantu $k > 1$.

Úloha 2 (A teď oboustranně a se smrštěním)

Zatím jsme přidávali jenom na konec pole. Šlo by konstrukci upravit tak, abychom mohli přidávat i na začátek a zachovali jsme přitom složitost?

A co kdybychom chtěli prvky z konce (či začátku) mazat?

Úloha 3 (Binární čítač)

Připomeňte si důkaz, že binární počítadlo (na začátku nastavené na 0) má amortizovanou konstantní složitost přičítání jedničky. Jak se změní amortizovaná složitost, pokud povolíme přičítání i odčítání jedničky? Lze upravit počítadlo, aby zůstala konstantní?

Bonusové úlohy

Úloha 4 (Binární čítač II)

Jsme ve stejné situaci jako v předchozí úloze (Binární čítač) a chceme jenom přičítat. Ukažte, že když za přehození k -tého bitu platíme 2^k (a nultý bit je nejméně významný bit), pak posloupnost ℓ přičtení jedničky stojí $\mathcal{O}(\ell \cdot n)$.

Úloha 5 (Sublineární množina)

Máme následující implementaci množiny (celých čísel), která podporuje operace INSERT, LOOKUP.

Množina M bude implementovaná jako pole polí, kde pole $M[i]$ má délku 2^i . Každé pole $M[i]$ je buď prázdné, nebo plné, a každé pole je seřazené (mezi jednotlivými poli nemusí být žádný vztah).

Konkrétně, máme-li v naší množině 9 prvků, tak pole $M[0]$, $M[3]$ budou plná a ostatní budou prázdná (vlastně to odpovídá binární reprezentaci čísla 10). Příkladem může být například:

```
M[0] = {5}
M[1] = {}
M[2] = {}
M[3] = {3, 4, 8, 10, 11, 15, 19, 22}
```

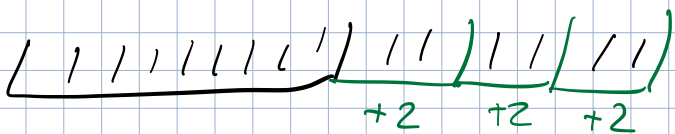
a) Jaká je složitost operace LOOKUP, kterou lze implementujeme tak, že na každém plném poli provedeme binární vyhledávání?

b) Operaci INSERT budeme implementovat následovně: začneme s tím, že vytvoříme prázdné pole s jedním prvkem, který vkládáme. Dále se podíváme na pole $M[0]$. Pokud je prázdné, vložíme naše nové pole tam. Pokud je plné, pak slijeme naše nové pole s polem $M[0]$ a pokračujeme dále s polem $M[1]$, dokud nenajdeme pole $M[i]$, které je prázdné. Jaká je (amortizovaná) složitost operace INSERT?

Hint: může se vám hodit předchozí úloha.

Co by se stalo, kdybychom místo zdvojnásobování kapacity C zvyšovali jinak?

1. $C \rightsquigarrow C + k$, kde $k \geq 1$ je konstanta,
2. $C \rightsquigarrow C^2$,
3. $C \rightsquigarrow k \cdot C$ pro konstantu $k > 1$.

1)  $\rightarrow O(n)$

2)  $\rightarrow O(1)$

3) Jako dyn. pole $\rightarrow O(1)$ $\rightarrow$ je to jen alternativní dyn. pole s $k=2$

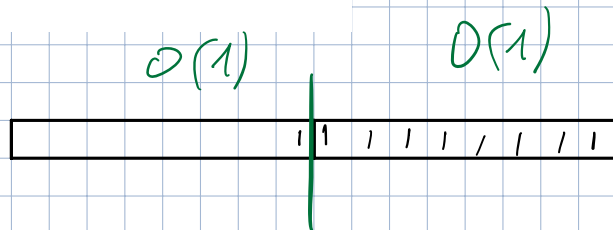
polože můžeme alokovat v $O(1)$, pak jsem schopný vnést peníze v $O(1)$ a vždy těch penězů budu mít více než u kl. dyn. pole, které je také $O(1)$.

peníze
reálné
počet
prvků

Úloha 2 (A teď oboustranně a se smrštěním)

Zatím jsme přidávali jenom na konec pole. Šlo by konstrukci upravit tak, abychom mohli přidávat i na začátek a zachovali jsme přitom složitost?

A co kdybychom chtěli prvky z konce (či začátku) mazat?



1

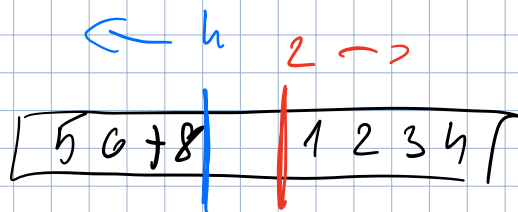
2 2

3 2 

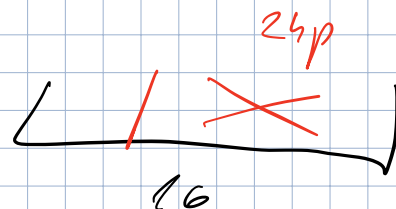
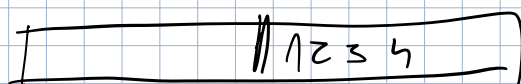
$\rightarrow$ jenom upravený lin. přístup polem



$\rightarrow$ alokace k.c., předplácím tři peníze.



$\rightarrow$ obvyklejší cyklická struktura



Úloha 3 (Binární čítač)

Připomeňte si důkaz, že binární počítadlo (na začátku nastavené na 0) má amortizovanou konstantní složitost přičítání jedničky. Jak se změní amortizovaná složitost, pokud povolíme přičítání i odčítání jedničky? Lze upravit počítadlo, aby zůstala konstantní?

$$T \leq A + (\Phi - \Phi')$$

1 1 1 1 1
↓
+
1 0 0 0 0 0
↓
-
1 1 1 1 1

Takle je problémový
scénář.
Můžeme ale vezt dva efektory,
oba pracující v $O(1)$
a na konci jen posadit součet.

1 0 0 1 1 3
↓

1 0 1 0 0 2

0 0 0 0 0 0

↓
0 0 0 0 0 1 1

1 1 1 1 1 n

↓
0 0 0 0 0 0